Scientific Research Publishing

Journal of Information Security, 2023, 14, 437-453
https://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

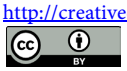
Design of a Cryptographic Algorithm in the Form of an API in Order to Secure Monetary Transactions in a Supermarket

Atsopmene Tango Vanette Eleonore[1], Gamom Ngounou Ewo Roland Christian[1], Kom Charles Hubert[1,2]

[1]Computer and Automatic Engineering Laboratory, Higher Normal School of Technical Education of the University of Douala, Douala, Cameroun
[2]Energy, Materials, Modeling and Methods Laboratory, National Polytechnic School of University of Douala, Douala, Cameroun
Email: vanytango@gmail.com

How to cite this paper: Eleonore, A.T.V., Christian, G.N.E.R. and Hubert, K.C. (2023) Design of a Cryptographic Algorithm in the Form of an API in Order to Secure Monetary Transactions in a Supermarket. *Journal of Information Security*, **14**, 437-453.
https://doi.org/10.4236/jis.2023.144024

Received: August 24, 2023
Accepted: October 22, 2023
Published: October 25, 2023



Abstract

Supermarkets and large-scale retail stores are usually subject to huge monetary transactions for certain customers' purchases. The computerization of these systems is common in supermarkets but the security of these transactions remains a mystery. This article presents an algorithm as an API based on symmetric cryptography that can enable end-to-end encryption of a monetary transaction in a supermarket. This algorithm is the first part of the complete supermarket management system which will be presented in the following article. The Python language and the Flask framework allow us to develop the algorithm as an independent component. Tests have been performed and our algorithm uses 98.49% less memory and 10.18% time saving than the AES algorithm.

Keywords

Application Programming Interface (API), Symmetric Cryptography, End-to-End Encryption

1. Introduction

Information security can be one of the main concerns of customers within an institution involving financial transactions or critical data. Indeed, users of money services face critical risks of intrusion into their accounts. Therefore, it is very important to build a system capable of certifying the identity of the sender and the recipient by a trusted third party who holds the identity certificates. In order to mitigate possible security vulnerabilities, many vendors have developed vari-

ous solutions in software systems [1] [2]. Security and privacy features of e-banking must be improved quickly to continue its growth. The use of electronic banking services has raised many concerns from different perspectives: government, businesses [3], banks [4], individuals and technology. It will therefore be interesting to set up security for data protection. The most suitable way to secure your data remains cryptography, as described in many books. F. J. Kherad, R. Naji, M. Malakooti and P. Haghighat in [5] develop an algorithm for securing electronic commerce transactions called FJ RC-4, the latter being derived from the RC4 symmetric key algorithm. In cryptographic form, these same algorithms are implemented in sm3 type financial smart cards for financial security [6]. These works as well as [5] [7] [8] [9] have inspired us to be able to implement an algorithm developed in the form of an API for securing data in electronic transactions in general and in supermarkets in particular. This article has five parts: the first presents general information about APIs; the second part presents the choice of the crypto system; the third presents the implementation of the algorithm; the fourth presents the results of the discussion and the performance of the algorithm; and the last part presents the technology and the API in question.

2. API General

Application Programming Interfaces (APIs) [10] [11] [12], a set of rules for how applications connect and communicate, provide frameworks for developers to create HTTP-based services accessible by software applications. The current development of Web APIs tends towards the architectural style Representational State Transfer (REST) [13] [14], which offers a high level of flexibility. The RESTful API is a software design pattern that specifies a uniform, predefined collection of stateless operations.

The REST API is a set of functions, rules, commands and protocols that contain general standards on how to exchange information between client and server [15]. An API exposes a set of data and functions to facilitate interactions between computer programs and allow them to exchange information [16].

A REST API will allow in our case to make our algorithm accessible to any monetary transaction security system. The structure of our system is shown in **Figure 1**:

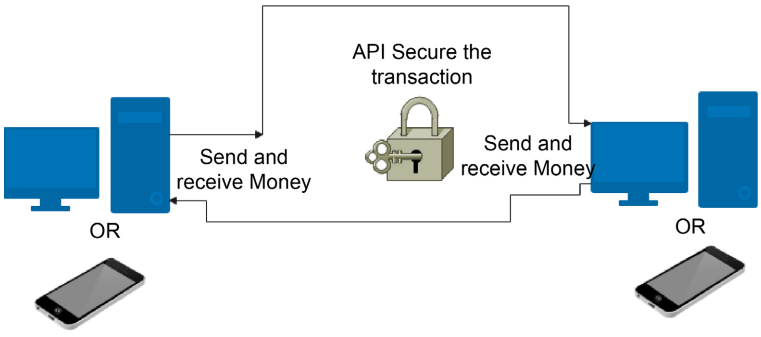


Figure 1. Fonctionning and role of API.

Figure 1 describes the operation and role of our API which provides end-to-end encryption in a monetary transaction between two entities which can be two computers, two telephones or even a server and a client. The algorithm will therefore have to retrieve parameters via the API URL and perform encryption and decryption just before the end of the monetary transaction.

3. Context

In Cameroon and in the CEMAC zone in general, we observe that many buyers in supermarkets use banknotes and coins to pay their expenses. This generates some problems, namely: reimbursement problems for lack of change, queuing problems in front of the few cash desks available for the payment of purchases. Society being more and more evolving, it is quite obvious to observe a considerable evolution of the NTIC which articulate with the payments and payments of invoices since a telephone or any terminal. Despite this growing evolution in payment technology, we still see people paying their bills with notes and coins, as is usually the case in supermarkets.

Faced with the problems mentioned above, we offer an automatic multipoint order acquisition system on behalf of each user. The security of accounts and transactions is therefore to be discussed in this article. A customer can therefore open an account and leave his money in the account in case of lack of money.

4. Choice of the Type of Cryptosystem

From the start in this research work on securing monetary transactions in the context of a supermarket, we first retained the cryptography method used. After a detailed study of cryptography systems, we opted for symmetric key cryptography [17] [18] because of its simplicity, speed and the existence of a unique key allowing encryption and decryption thanks to this key.

This type of cryptography is based on simple mathematical functions such as substitution and permutation [19]. There are therefore two types of symmetric ciphers namely the stream cipher and the block decipher [20] [21].

In contrast, asymmetric encryption uses two separate keys: a public key for encryption and a private key for decryption. The public key is known to everyone and can be used by anyone to encrypt data. The private key, on the other hand, is kept secret by its owner and is used to decrypt the data encrypted using the corresponding public key. This approach solves the problem of secure key distribution, because only public keys are shared.

Asymmetric encryption is generally slower than symmetric encryption due to the mathematical complexity involved, hence its choice as it is easy to implement and less cumbersome than asymmetric encryption.

Choice of Symmetric Encryption Type

For the implementation of our algorithm, we opted for block encryption which

consists of cutting a message or a set of characters into blocks of fixed sizes then encrypting each block using a public key and all the blocks assembled thus forming a cryptogram. The cipher block is more explicit in Figure 2.

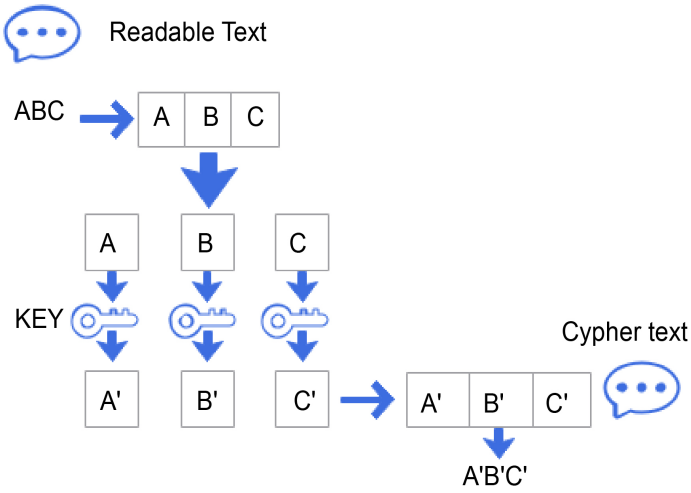


Figure 2. Principle of operation of block cipher.

After having presented our main choices, we can now present the architecture of the system.

5. Algorithm Implementation

The first step was to implement the different equivalence tables or substitution tables.

5.1. Block Substitution Table

Since we are implementing a block cipher, each entry of the amount of a transaction is an integer and is distributed in an array of 12 cells, ranging from index 0 to index 11. Each square in the array has the table has first of all a combination of two characters in the end. The combination game was done as follows:

- We create a table containing 12 columns and 5 rows. The first line represents the indices of the array.
- At the level of the second line of the table, we fill the boxes of this table with the letters of the French alphabet in the interval {a…….l} we continue with this filling at the level of the fourth line following the interval {m…….x}.
- At the level of the third line of the table we fill the boxes of this table with the letters of the French alphabet in the interval {z…….x} we continue with this filling at the level of the fifth line following the interval {n……c}.

By applying these equivalences we obtained Table 1 which summarizes the substitutions that we have made.

Table 1 was reduced by a selection of two characters as shown in Table 2. The process is explained below:

Table 1. First table of equivalence of the indices of a block table.

0	1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	f	g	h	i	j	k	l
z	y	x	w	v	u	t	s	r	q	p	o
m	n	o	p	q	r	s	t	u	v	w	x
n	m	l	k	j	i	h	g	f	e	d	c

Table 2. Reduction of the first equivalence table.

0	1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	f	g	h	i	j	k	l
z	y	x	w	v	u	t	s	r	q	p	o
m	n	o	p	q	r	s	t	u	v	w	x
n	m	l	k	j	i	h	g	f	e	d	c

The reduction of the elements of the initial array was done in the form of stairs. And so we reduce this index equivalence table to much simpler. Table 3 presents the final equivalences.

Table 3. Final equivalence table.

0	1	2	3	4	5	6	7	8	9	10	11
a	y	o	d	v	r	g	s	u	j	p	X
z	n	l	w	q	i	t	t	F	q	w	c

Table 3 represents the equivalences according to the indices of the table which must contain a number at each index. By inserting an element into the array, the array will already have content. In our case, we can only insert one digit per box in the table a digit will be considered as a block.

5.2. Digit Substitutions

Securing a transaction is equivalent to securing any number, so we will manipulate numbers in the range {0……9}. The following table represents the equivalences between the different numbers.

According to the above, the number zero will be replaced by "aq", the number 1 will be replaced by "zs". A table cell can therefore contain 04 letters depending on the number inserted and the cell that contains the number. A table cell must therefore contain 32 bits in total because each letter is coded on 8 bits.

5.3. Logic Programming

We developed our algorithm in two functions namely the encryption function

and the decryption function. We present the flowchart of the algorithm and its implementation with the PYTHON language.

5.3.1. Organization Chart

Figure 3 shows the encryption and decryption flowchart. It all starts with retrieving the number as plain text, then making substitutions from an array based on the values of the array elements shown in Table 4; then concatenate each cell element with the equivalent cell by cell as shown in Table 3 then do the inverse of this table. We have segmented the replaced text into blocks, *i.e.* one box in the table equals one block. On each block, the secret key is applied using an XOR operator and the table is converted into text; At this point we have the ciphertext that marks the end of the cipher function. Once the ciphertext is obtained, we move on to the decryption process, which begins with converting the ciphertext into a table; The different blocks are associated with the key with an XOR operator then we do the inverse of this table. After the reverse, the values are compared with those of Table 3 and Table 4 of equivalence. The table is transformed into plain text and finally into a number thus marking the end of the decryption function.

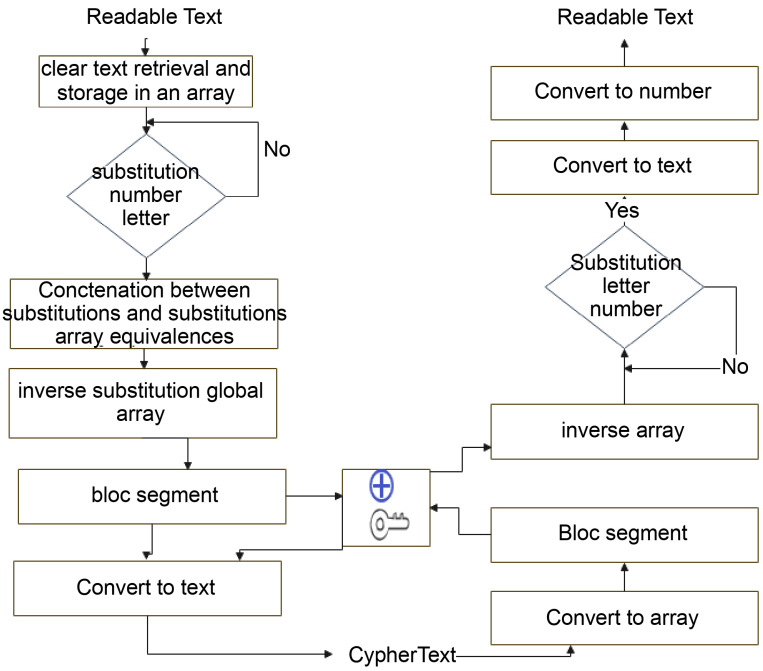


Figure 3. Organizational chart.

Table 4. Digit equivalence table.

0	1	2	3	4	5	6	7	8	9
a	z	e	r	t	y	u	i	o	p
q	s	d	f	g	h	j	k	l	m

5.3.2. The Encryption Function

This function consists in encrypting the message according to the Key. It therefore takes two parameters, namely the key and the amount or number to encrypt. This function is written in several steps namely.

Recovery of the amount intended for encryption: In this operation, the data inserted into a parameter of the function is recovered as shown in **Figure 4.** The recovered data will then be transformed into a table as shown in **Figure 5** (by the instruction cast in PYTHON) where each digit is stored in a table cell. We therefore recover this array in a variable and each time we test the length of the array (using the PYTHON len() function) to restructure the values in our 12-entry equivalence table. The assigned values undergo a CAST (by the (int) instruction in front of the variable name) in order to receive only the integers as shown in **Figure 6.** This entire process is represented by the figures below.

```
def encrypt(number,key):
```

Figure 4. Start of the encryption function.

```
y = list(number)
```

Figure 5. Transformation of the variable into an array of one character per cell.

```
elif(len(y) ==
    2):
        x0 = 0
        x1 = 0
        x2 = 0
        x3 = 0
        x4 = 0
        x5 = 0
        x6 = 0
        x7 = 0
        x8 = 0
        x9 = 0
        x10 =
  int(y[0])
        x11 =
  int(y[1])
```

```
elif (len(y) ==
      3):
          x0 = 0
          x1 = 0
          x2 = 0
          x3 = 0
          x4 = 0
          x5 = 0
          x6 = 0
          x7 = 0
          x8 = 0
           x9 =
   int(y[0])
          x10 =
   int(y[1])
          x11 =
   int(y[2])
```

```
if(len(y
) == 1):
 x0 = 0
 x1 = 0
 x2 = 0
 x3 = 0
 x4 = 0
 x5 = 0
 x6 = 0
 x7 = 0
 x8 = 0
 x9 = 0
x10 = 0
 x11 =
int(y[0]
   )
```

```
elif (len(y) == 12):
        x0 = int(y[0])
        x1 = int(y[1])
        x2 = int(y[2])
        x3 = int(y[3])
        x4 = int(y[4])
        x5 = int(y[5])
        x6 = int(y[6])
        x7 = int(y[7])
        x8 = int(y[8])
        x9 = int(y[9])
           x10 =
   int(y[10])
           x11 =
   int(y[11])
```

Figure 6. CAST and recovery of variables one by one in nb variables just before substitution.

Determine the equivalences according to the figures inserted: this part consists in giving the equivalence of the table of substitution of the figures according to the content of the variables x (x0… x11) we put this new content in the same variables then make a concatenation with the cells of the array that already contains substitution values. These steps are represented by the code snippets in **Figure 7**.

```
#first number
   if (x0==0):
     subs0="aq"
elif(x0==1):
    subs0 = "zs"
elif(x0==2):
    subs0 = "ed"
elif(x0==3):
    subs0 = "rf"
```

```
#second number
    if (x1==0):
        subs1="aq"
elif(x1==1):
        subs1 = "zs"
elif(x1==2):
        subs1 = "ed"
elif(x1==3):
        subs1 = "rf"
```

```
#tirth number
    if (x2 == 0):
        subs2     =
"aq"
elif (x2 == 1):
        subs0     =
"zs"
elif (x2 == 2):
        subs2     =
"ed"
elif (x2 == 3):
```

Figure 7. Digit substitution.

Once the substitution has been made, that is to say establishing the same process from the variable x0 to the variable x11, we can therefore associate them with the array already containing some substitution elements. These steps are represented by the code snippets in **Figure 8**.

```
substitu-
tion=["az"+subs0,"yn"+subs1,"ol"+subs2,"dw"+subs3,"vq"+subs4,"ri"+su
bs5,"gt"+subs6,"st"+subs7,"uf"+subs8,"jg"+subs9,"pw"+subs10,"xc"+sub
s11]
```

Figure 8. Final substitution table.

Inverse of the table and combination with the key: This part allows you to invert (with the function reverse ( )) the table then to combine each cell with the-key. The excerpt from **Figure 9** shows an illustration of this.

```
substitution.reverse()
```

Figure 9. Inverse of the final substitution table.

After the inverse we associate each cell of the inversed array to the key with an XOR (^). The excerpt from **Figure 10** illustrates this.

```
cryptogram0 = xor_strings(substitution[0],key)
 cryptogram1 = xor_strings(substitution[1],key)
 cryptogram2 = xor_strings(substitution[2],key)
 cryptogram3 = xor_strings(substitution[3],key)
 cryptogram4 = xor_strings(substitution[4],key)
 cryptogram5 = xor_strings(substitution[5],key)
 cryptogram6 = xor_strings(substitution[6],key)
 cryptogram7 = xor_strings(substitution[7],key)
```

Figure 10. Association of array elements with key.

The xor_strings() function allowed us to associate the substitutions to the key

with the XOR operator, its definition is presented in Figure 11.

```
def xor_strings(str1, str2):
        result = ""
        for char1, char2 in zip(str1, str2):
     xor_value = ord(char1) ^ ord(char2)
             result += chr(xor_value)
        return result
```

Figure 11. Function for associating an element with the key.

Obtaining the cryptogram or ciphertext: After the association with the public key, a table of encrypted data is brought out and then transformed into a string (with the join() function). Finally we made a Cast to transform to have the hexadecimal value of the cryptogram. This is shown in Figure 12.

```
crypto-
gram=[cryptogram0,cryptogram1,cryptogram2,cryptogram3,cry
pto-
gram4,cryptogram5,cryptogram6,cryptogram7,cryptogram8,cry
ptogram9,cryptogram10,cryptogram11]
final = ' '.join(cryptogram)
finalcryptogram = final.encode().hex()
```

Figure 12. Final cryptogram.

We therefore obtain an encrypted text contained in a variable and which can be saved in the database while being encrypted. So we moved on to the decryption function.

5.3.3. Decryption Function

This function allows you to decrypt cipher text with the same key used during encryption. It therefore takes two parameters, namely the ciphertext and the key. Here are the steps of this function.

Cast of the hexadecimal type into a character string: This step makes it possible to transform the type of the cryptogram into a character string by Figure 13.

```
description = hex_to_string(finalcryptogram)
```

Figure 13. Hex to string transformation.

Transformation of the cryptogram into an array: This operation was carried out with the function string_to_list() written by us which allows the transformation of the cryptogram into an array of 4 elements per cell. Consider a table with one row and 12 columns. This operation is illustrated by the code in Figure 14.

```
decryptTable = string_to_list(description,4)
```

Figure 14. Transformation of the final cryptogram into a table.

The string_to_list() function which allows us to transform a string into an ar-

ray with a specific number of elements is defined by Figure 15.

```
def string_to_list(chaine, cases_number):
liste = []
    for i in range(0, len(chaine), cases_number):
sous_chaine = chaine[i:i+ cases_number]
liste.append(sous_chaine)
    return liste
```

Figure 15. Function for transforming a string into an array of elements.

Combination with key and array inverse: This part is used to associate each element contained in the previous array with the key with the XOR (^) operator. Then we will do the reverse of this array. The code snippets in Figure 16 illustrate this.

```
decryptTableElement0= xor_strings(decryptTable[0],key)
 decryptTableElement1 = xor_strings(decryptTable[1],key)
 decryptTableElement2 = xor_strings(decryptTable[2],key)
 decryptTableElement3 = xor_strings(decryptTable[3],key)
 decryptTableElement4 = xor_strings(decryptTable[4],key)
 decryptTableElement5 = xor_strings(decryptTable[5],key)
```

Figure 16. Association of each element of the cryptogram with the key.

The final table after this operation is shown in Figure 17.

```
decrypTableFin-
al=[decryptTableElement0,decryptTableElement1,decryptTableElement2,dec
ryptTableEle-
ment3,decryptTableElement4,decryptTableElement5,decryptTableElement6,
decryptTableEle-
ment7,decryptTableElement8,decryptTableElement9,decryptTableElement10,
decryptTableElement11]
```

Figure 17. Decryption table.

Once this operation is complete, we apply the inverse function (reverse()) to the table of Figure 16, then we look for the equivalences based on the different tables of equivalences. Code snippets illustrate this in Figure 18.

```
if(decrypTableFinal[0]=='azaq'
):
        decryptNumber0= '0'
elif(decrypTableFinal[0]=='azz
s'):
        decryptNumber0 = '1'
elif  (decrypTableFinal[0]  ==
'azed'):
        decryptNumber0 = '2'
elif  (decrypTableFinal[0]  ==
'azrf'):
        decryptNumber0 = '3'
elif  (decrypTableFinal[0]  ==
'aztg'):
        decryptNumber0 = '4'
```

```
#Second number
if (decrypTableFinal[1] == 'ynaq'):
        decryptNumber1 = '0'
elif (decrypTableFinal[1] == 'ynzs'):
        decryptNumber1 = '1'
elif (decrypTableFinal[1] == 'yned'):
        decryptNumber1 = '2'
elif (decrypTableFinal[1] == 'ynrf'):
        decryptNumber1 = '3'
elif (decrypTableFinal[1] == 'yntg'):
        decryptNumber1 = '4'
```

Figure 18. Equivalences for decryption.

Bring out the table the final table of the clear number: We just associate the equivalence variables of Figure 18 in a table of character strings. Figure 19 illustrates this.

```
FinalDecryptTa-
ble=[decryptNumber0,decryptNumber1,decryptNumber2,decryptNumber3,
decryptNumb-
er4,decryptNumber5,decryptNumber6,decryptNumber7,decryptNumber8,
decryptNumber9, decryptNumber10,decryptNumber11]
```

Figure 19. Plain text output table.

Thereafter we transform the array of Figure 18 into a character string and then into an integer, illustrated in Figure 20.

```
FinalDecryptTableString = ' '.join(FinalDecryptTable)
FinalDecryptTableInt = int(FinalDecryptTableString)
```

Figure 20. Plain text output.

We have above the last step marking the end of the encryption and decryption of a number via the algorithm presented above. And we will move on to the various results, discussions and performance test of our algorithm.

6. Results, Discussions and Performance

6.1. Results and Discussions

We will do a test with a random number and a random key, the plain text is: 123456, the key is: ab34. We run the algorithm and we have the result following the steps stated above. This is shown in Figure 21.

Figure 21. Running the program.

The program executed following the steps listed earlier in this article, we will look at the cryptogram presented in Figure 22.

```
1901465e11154a5c0b0547531404415212165650061649471 30b5245171
352450 51552450e0e5245180c524500185245
```

Figure 22. Cryptogram.

This ciphertext consists of 96 fixed length characters since one character repre-

sents 4 bits, the ciphertext represents 384 bits in general. We can therefore discuss and make the following hypotheses:

- The cryptogram present in **Figure 21** is impossible to decipher by brute force, Attack by collision, Attack by plaintext chosen but can be vulnerable to Attack by dictionary because the key is held by the user and this key is generally a password that someone close to him can imagine.
- The cryptogram is not heavy and does not require database robustness. It can be stored in an online or local database.
- The ciphertext above can be maintained during processing, ie encrypt at the start and decrypt at the end. The use of the decryption function with the key makes it possible to have the clear text, and that of encryption with the same key leads us to this ciphertext, the use of end-to-end encryption is thus possible.
- The encryption function can be independent of the decryption function. the only common point is the key which must be unique in both cases. For this purpose, the customer may have the possibility to encrypt his amount or his transaction and to decrypt it, he is therefore responsible for his own security.
- In terms of performance, the execution of the encryption and decryption functions is estimated in microseconds by functions provided by the PYTHON language documentation.

6.2. Performance

We were inspired by [18] [22] to present the performance of our algorithm, having the concern to use a more recent AES library with a test on the same machine.

6.2.1. Performance of the Machine Used for the Test

The personal computer used in all programs and experiments was a 4th generation Intel Core i 4300M, 2.60 GHz processor with 8 GB of RAM and 1 TB of hard disk capacity. The performance of this algorithm is evaluated on the basis of parameters such as required memory and simulation time. We therefore used the php library of [23] an interesting and fairly recent library to do our comparative tests.

6.2.2. Comparison According to the Memory and Execution Time Required to Use the Algorithms

We did a test of the library and our algorithm, the value to encrypt is 1234 and the password is 1234 for both algorithms. The PYTHON functions that allowed us to evaluate the memory needed and the execution time of our algorithm are the sys.getsizeof () and time.perf_counter () functions respectively. **Table 5** presents the comparison in terms of performance. We can materialize these performances in graphicform as shown in **Figure 23**.

We can have this graph showing the performance of these algorithms.

This curve presents a downward evolution from the DES algorithm, through AES to our algorithm in terms of required memory and simulation time. Thus by using our algorithm we gain in performance which is expressed by the re-

quired memory (*i.e.* is 6.3 KB) and the simulation time ($1.9*10^{-5}$ seconds).

Table 5. Performance comparison and overview.

Technical	Memory Required for Implementation (KB)	Simulation Temps (second)
AES(HEX)	9.816	$2.2*10^{-4}$
Notre algorithm (HEX)	0.148	$1.7*10^{-5}$

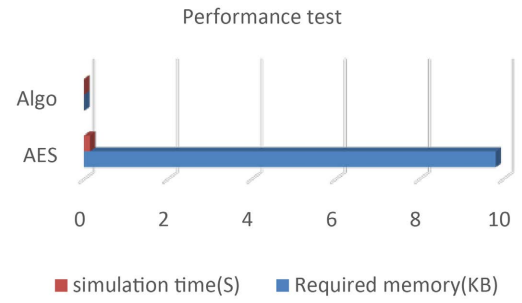


Figure 23. Performance overview graph.

After the performance graph of our algorithms compared to AES, let's do the comparison based on the cryptogram structure.

6.2.3. Comparison Based on Cryptogram Structure

The different cryptograms are shown in **Table 6**.

Table 6. Comparison of cryptograms.

Method	Cryptogram example
AES (HEX)	52282ce90391156b787f47e4cdb2876a
Algorithm (HEX)	49514753414541525b5556504454494 74246524556465245435b5245474352 45554552455e5e5245485c524550485245

Table 6 shows us the comparison between our cryptogram with readable text 1234 and password 1234, and the AES cryptogram with readable text 1234 and password 1234. We notice that our cryptogram is more robust and much heavier than that of the AES. This conclusion presents our reason for implementing the algorithm in question to gain performance while allowing a more robust cryptogram.

7. API

7.1. Overview of Routes

To highlight our algorithm we divided it into two functions namely the encryption function (encrypt) and the decryption function (decrypt) thereafter we used the Flask framework to have links to each function which constitute the API.

The flask framework allowed us to determine the API routes which are presented in Figure 24.

```
@app.route('/encrypt/<x>/<key>')
def encrypt(x,key):
```

Figure 24. Route to the encrypt function.

Figure 24 shows the routes to the encrypt() function which encrypts the number "x" indicated as a parameter using the key "key". At the end of this function we have a cryptogram. With the encrypt() function thus presented we move on to the route of the decrypt function shown in Figure 25.

```
@app.route('/decrypt/<finalcryptogram>/<key>')
def decrypt(finalcryptogram, key):
```

Figure 25. Route to the decrypt function.

This route presents the decrypt function taking as parameter the cryptogram and the key to find the initial cipher.

7.2. Testing with Postman

We performed the Tests with Postman function by function. The encrypt function gave us the results shown in the following figure.

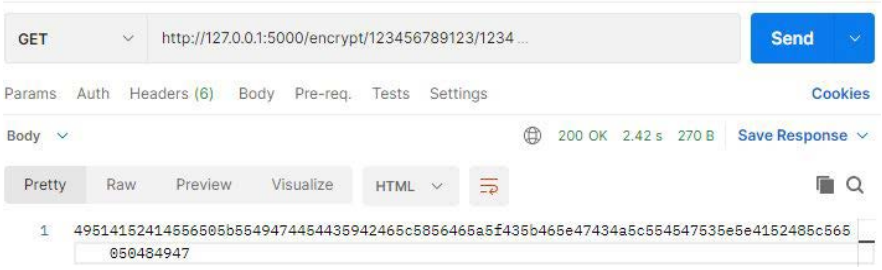


Figure 26. API test, encrypt function.

Figure 26 presents the encrypt function with for x = 123456789123 and for key = 1234 which has been encrypted and we have a cryptogram. The case of the decrypt function is shown in Figure 27.

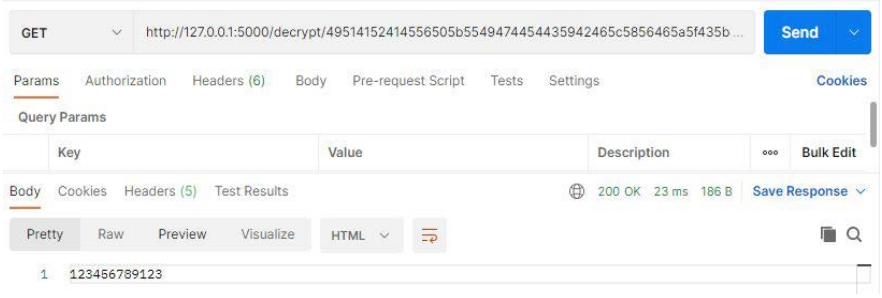


Figure 27. API test, decrypt function.

Figure 27 shows the decryption in the decrypt function taking as parameter the previous cryptogram and the key and we have in output a clear number or amount.

We have therefore output well after a collection of the API thus bringing together the two functions in a JSON format in Figure 28.


```
{
    "info": {
        "_postman_id": "f0ef4839-0267-4810-8289-390d96232e83",
        "name": "algo",
        "schema":
"https://schema.getpostman.com/json/collection/v2.0.0/collection.json"
    },
    "item": [
        {
            "name": "ecrypt",
            "request": {
                "method": "GET",
                "header": [],
                "url": "http://127.0.0.1:5000/encrypt/123456789123/1234"
            },
            "response": []
        },
        {
            "name": "decrypt",
            "request": {
                "method": "GET",
                "header": [],
                "url":
"http://127.0.0.1:5000/decrypt/49514152414556505b55494744544359424 65c5856465
a5f435b465e47434a5c554547535e5e4152485c565050484947/1234"
            },
            "response": []
        }
    ]
}
```


Figure 28. API collection in JSON format.

8. Conclusion

In a framework defined as that of shopping malls, the security of transactions represents a fundamental process to guarantee the confidentiality of transactions and the retention of data which is generally of a financial nature. The implemented algorithm presents security based on substitution and permutation techniques. This rather interesting algorithm has the robustness to make the system very effective in imperviousness to attacks while making the customer sovereign of his own security of payments in general and payments in a supermarket in particular. Our algorithm uses different cryptographic techniques, namely confidentiality, integrity, authentication and non-repudiation. Also, the state of payment systems today, and fundamental security requirements for secure payment have been our priority in this article while giving the constitution of our algorithm and its advantages. The article presented the first part which is the security algorithm transactions in a supermarket which will be useful when implementing the system. And it will be the subject of the second article.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Li, D. and Yang, Y. (2012) Enhance Value by Building Trustworthy Software-Reliant System of Systems from Software Product Lines. 2012 *Third International Workshop on Product LinE Approaches in Software Engineering* (*PLEASE*), Zurich, 04-04 June 2012, 13-16. https://doi.org/10.1109/PLEASE.2012.6229761

[2] Chess, B. and Arkin, B. (2011) Software Security in Practice. *IEEE Security & Privacy*, **9**, 89-92. https://doi.org/10.1109/MSP.2011.40

[3] Joukov, N., Shorokhov, V. and Tantsuyev, D. (2014) Security Audit of Data Flows across Enterprise Systems and Networks. *The* 9*th International Conference for Internet Technology and Secured Transactions* (*ICITST*-2014), London, 08-10 December 2014, 240-247. https://doi.org/10.1109/ICITST.2014.7038813

[4] Khande, R. and Patil, Y. (2014) Online Banking in India: Attacks and Preventive Measures to Minimize Risk. *International Conference on Information Communication and Embedded Systems* (*ICICES*2014), Chennai, 27-28 February 2014, 1-5. https://doi.org/10.1109/ICICES.2014.7033940

[5] Kherad, F.J., Naji, H.R., Malakooti, M.V. and Haghighat, P. (2010) A New Symmetric Cryptography Algorithm to Secure E-Commerce Transactions. 2010 *International Conference on Financial Theory and Engineering*, Dubai, United Arab Emirates, 18-20 June 2010, 234-237. https://doi.org/10.1109/ICFTE.2010.5499388

[6] Hu, Y., Wu, L., Wang, A. and Wang, B. (2014) Hardware Design and Implementation of SM3 Hash Algorithm for Financial IC Card. 2014 *Tenth International Conference on Computational Intelligence and Security*, Kunming, China, 15-16 November 2014, 514-518. https://doi.org/10.1109/CIS.2014.176

[7] Ofosu, A.E., Kester, Q.-A. and Anyanewah, A.J.A. (2019) A Cryptographic Algorithm Based on Aes Cipher Andnondeterministic Algorithm Approach for Key Generation. 2019 *International Conference on Computing, Computational Modelling and Applications* (*ICCMA*), Coast, 27-29 March 2019, 105-1054. https://doi.org/10.1109/ICCMA.2019.00024

[8] Upadhyay, D., Zaman, M., Joshi, R. and Sampalli, S. (2022) An Efficient Key Management and Multi-Layered Security Framework for SCADA Systems. *IEEE Transactions on Network and Service Management*, **19**, 642-660. https://doi.org/10.1109/TNSM.2021.3104531

[9] Chiba, Z., Abghour, N., Moussaid, K., Omri, A.E. and Rida, M. (2018) A Hybrid Optimization Framework Based on Genetic Algorithm and Simulated Annealing Algorithm to Enhance Performance of Anomaly Network Intrusion Detection System Based on BP Neural Network. 2018 *International Symposium on Advanced Electrical and Communication Technologies* (*ISAECT*), Rabat, 21-23 November 2018, 1-6. https://doi.org/10.1109/ISAECT.2018.8618804

[10] Jones, G., *et al*. (2022) API Development Increases Access to Shared Computing Resources at Boston University. *Journal of Software Engineering and Applications*, **15**, 197-207. https://doi.org/10.4236/jsea.2022.156011

[11] Hassan, B., Namir, K., Rachiq, A., Elhoussin, L. and Fouzia, B. (2018) MapReduce Programs Simplification Using a Query Criteria API. *International Journal of Advanced Computer Science and Applications*, **9**. https://doi.org/10.14569/IJACSA.2018.090607

[12] Almotiri, S., Alosaimi, N. and Abdullah, B. (2021) Using API with Logistic Regression Model to Predict Hotel Reservation Cancellation by Detecting the Cancellation Factors. *International Journal of Advanced Computer Science and Applications*, **12**. https://doi.org/10.14569/IJACSA.2021.0120688

[13] Vinoski, S. (2007) REST Eye for the SOA Guy. *IEEE Internet Computing*, **11**, 82-84. https://doi.org/10.1109/MIC.2007.22

[14] Khare, R. and Taylor, R.N. (2004) Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems. *Proceedings*. 26*th International Conference on Software Engineering*, Edinburgh, UK, 28 May 2004, 428-437. https://doi.org/10.1109/ICSE.2004.1317465

[15] Ignatius Moses Setiadi, D.R., Faishal Najib, A., Rachmawanto, E.H., Atika Sari, C., Sarker, K. and Rijati, N. (2019) A Comparative Study MD5 and SHA1 Algorithms to Encrypt REST API Authentication on Mobile-based Application. 2019 *International Conference on Information and Communications Technology* (*ICOIACT*), Yogyakarta, Indonesia, 24-25 July 2019, 206-211. https://doi.org/10.1109/ICOIACT46704.2019.8938570

[16] Masséand, M.H. and Massé, M. (2012) REST API Design Rulebook: Designing Consistent Restful Web Service Interfaces. https://www.oreilly.com/library/view/rest-api-design/9781449317904/

[17] Chandra, S., Paira, S., Alam, S.S. and Sanyal, G. (2014) A comparative survey of Symmetric and Asymmetric Key Cryptography. 2014 *International Conference on Electronics*, *Communication and Computational Engineering* (*ICECCE*), Hosur, Tamilnadu, India, 17-18 November 2014, 83-93. https://doi.org/10.1109/ICECCE.2014.7086640

[18] Shao, F., Chang, Z. and Zhang, Y. (2010) AES Encryption Algorithm Based on the High Performance Computing of GPU. 2010 *Second International Conference on Communication Software and Networks*, Singapore, 26-28 February 2010, 588-590. https://doi.org/10.1109/ICCSN.2010.124

[19] Radwan, A.G., AbdElHaleem, S.H. and Abd-El-Hafiz, S.K. (2016) Image Encryption Algorithms Using Non-Chaotic Substitutions and Permutations. 2016 13*th International Conference on Electrical Engineering/Electronics*, *Computer*, *Telecommunications and Information Technology* (*ECTI-CON*), Chiang Mai, Thailand, 28 June-01 July 2016, 1-6. https://doi.org/10.1109/ECTICon.2016.7561281

[20] Chunguang, H., Hai, C., Yu, S. and Qun, D. (2015) Permutation of Image Encryption System Based on Block Cipher and Stream Cipher Encryption Algorithm. 2015 *Third International Conference on Robot*, *Vision and Signal Processing* (*RVSP*), Kaohsiung, Taiwan, 18-20 November 2015, 163-166. https://doi.org/10.1109/RVSP.2015.46

[21] Ismoyo, D.D. and Wardhani, R.W. (2016) Block Cipher and Stream Cipher Algorithm Performance Comparison in a Personal VPN Gateway. 2016 *International Seminar on Application for Technology of Information and Communication* (*ISemantic*), Semarang, Indonesia, 05-06 August 2016, 207-210. https://doi.org/10.1109/ISEMANTIC.2016.7873839

[22] Mandal, A.K., Parakash, C. and Tiwari, A. (2012) Performance Evaluation of Cryptographic Algorithms: DES and AES. 2012 *IEEE Students' Conference on Electrical*, *Electronics and Computer Science*, Bhopal, India, 01-02 March 2012, 1-5. https://doi.org/10.1109/SCEECS.2012.6184991

[23] Eigelsreiter, R. (2022) CryptoJs 3.x AES Encryption/Decryption on Client Side with Javascript and on Server Side with PHP. https://packagist.org/packages/brainfoolong/cryptojs-aes-php